There are three main classes that you need to know about if you use the upf code for automatic uncertainty propagation.

```
namespace upf {
  class iestimate;
  class oestimate;
  class set_r;
}
```

The `iestimate` class represents "input estimates," or estimated values for input quantities in a measurement function. The `oestimate` class represents "output estimates," or calculated values of output quantities in a measurement function. The `set_r` class is used to assign correlation coefficients to pairs of input estimates.

It's probably also good to realize that both `iestimate` and `oestimate` are derived from an abstract base class called `estimate`, which defines the underlying structure.

You can construct an `iestimate` from a numerical value and a standard uncertainty, with optional degrees of freedom. You can construct an `oestimate` from a numerical value (no uncertainty), from an `iestimate`, or from another `oestimate`. There are also constructors that are used to combine sensitivity coefficients when calculating with estimates and propagating uncertainty.

```
iestimate::iestimate();
iestimate::iestimate(const iestimate& X);
explicit iestimate::iestimate(double x, double u=0.0, double df=infinity);

oestimate::oestimate();
oestimate::oestimate(double y);
oestimate::oestimate(double y, double s);
oestimate::oestimate(const estimate& X);
oestimate::oestimate(const oestimate& X);
oestimate::oestimate(double val, const estimate& X, double dFdX);
oestimate::oestimate(double val,
     const estimate& X, double dFdX,
     const estimate& Y, double dFdY);
oestimate::oestimate(double val,
     const estimate& X, double dFdX,
     const estimate& Y, double dFdY,
     const estimate& Z, double dFdZ);
```

The last three constructors are used in the functions provided in the upf math library:

pow	fabs	sqrt	exp	log	log10	sin	cos	tan	asin
acos	atan	atan2	sinh	cosh	tanh	asinh	acosh	atanh	hypot
expm1	zpdf	zcdf	ztail	zcritical	tpdf	tcdf	ttail	tcritical	ddc

There are a few functions that return iestimates based on parametrized distributions.

```
iestimate Gauss(double m, double s);           // normal distribution
iestimate Rect(double m, double a);            // rectangular distr.
iestimate Tri(double m, double a);             // triangular distr.
iestimate Trap(double m, double a, double b);  // trapezoidal distr.
iestimate U(double m, double a);               // U distr.
iestimate Poi(double m);                       // Poisson distr.
iestimate Poi1(double m);                      // (from MARLAP)
```

The `estimate` class has member functions that return values, uncertainties, and degrees of freedom.

```
double estimate::value() const;      // numerical value
double estimate::u()const;           // standard uncertainty
double estimate::ur()const;          // relative standard uncertainty
double estimate::u2()const;          // squared standard uncertainty
double estimate::ur2()const;         // squared relative standard uncertainty
double estimate::df() const;         // degrees of freedom
double estimate::U(double k=2.0) const;      // expanded uncertainty
double estimate::Ur(double k=2.0) const;     // relative expanded unc.
double estimate::U_p(double p=0.95) const;   // expanded unc., specified p
double estimate::Ur_p(double p=0.95) const;  // relative
```

There are **`static`** member functions for covariance and correlation coefficients.

```
static double estimate::u(const estimate& X, const estimate& Y);
static double estimate::r(const estimate& X, const estimate& Y);
```

The `estimate` class has other **`static`** functions that affect output formatting.

```
void     estimate::standard();              // select standard uncertainty
void     estimate::coverage(double k);      // expanded uncertainty, set k
double   estimate::coverage();              // return coverage factor k
void     estimate::confidence(double p);    // expanded uncertainty, set p
double   estimate::confidence();            // return confidence p
void     estimate::precision(unsigned n);   // set precision for uncertainty
unsigned estimate::precision();             // get precision
void     estimate::decimal_point(char dp); // set the decimal point char
char     estimate::decimal_point();         // return the decimal point char
void     estimate::grouping(bool b);        // turn digit grouping on or off
bool     estimate::grouping();              // return current setting
```

By default the decimal point character is set to `'\0'`, which tells the class to use the decimal point associated with the `std::locale` of the output stream. By default the precision is set to 2, causing uncertainties to be rounded to two figures. If you change the precision to 0, the class will use the precision associated with the output stream (set via `std::setprecision`).

There are also ostream "manipulators" that can perform similar functions.

```
str << upf::stdunc;
str << upf::coverage(k);
str << upf::confidence(p);
str << upf::precision(n);
str << upf::decimal_point(dp);
str << upf::grouped;
str << upf::ungrouped;
```

Note that these manipulators change settings for all ostreams, not just the specified stream `str`.

The output operator << determines whether to use fixed or scientific format by checking the format flags for the output stream. The `upf` namespace provides another manipulator `general`, which explicitly clears the stream format flags for both fixed and scientific format, resulting in the default format, which is sometimes called "general."

The estimate class also has three member functions that can return a std::string.

```
std::string estimate::str() const;
std::string estimate::str_f() const;
std::string estimate::str_s() const;
```

These functions use "general," "fixed," and "scientific" format, respectively.

In some cases the rounding rules may be inconsistent with fixed format. For example, 1234 ± 587 with precision 2 must be printed using scientific notation as (1.23 ± 0.59) e+01 rather than 1230 ± 590, because the latter could imply more precision than intended in the uncertainty. In these cases, even if fixed format has been chosen, scientific format will be used.

The oestimate class provides a member function that returns the value of a specified sensitivity coefficient.

```
double oestimate::d(const iestimate& x) const;
```

The `upf` namespace includes a nested namespace `aux`, which provides **`inline`** synonyms for `estimate`'s member functions `u()`, `u2()`, `ur()`, `ur2()`, and `r()`, and a synonym for `oestimate`'s `d()` function.

Example:

```
using namespace std;
using namespace upf;

iestimate a, b;
oestimate y;
double    m = 33.172;

a = iestimate(0.274, 0.012);
b = iestimate(0.00625, 0.00089);
set_r(a, b) = 0.83;

y = a * exp(-b*m);
estimate::precision(2);
cout << coverage(2.0) << y << endl;
cout << "value = " << y.value() << " u = " << aux::u(y) << endl;
```